新型 SQL 注入攻击的研究与防范①

赵 阳, 郭玉翠

(北京邮电大学 理学院, 北京 100876)

摘 要: 针对一种以 HTTP Headers 为途径的新型 SQL 注入攻击进行了深入研究. 通过分析具体的 SQL 注入实例, 揭示了该新型 SQL 注入攻击的原理, 并提出了针对此类攻击的防范手段. 通过 ip 过滤, 数据校验, 机器学习等手段建立了一套完整的防御模型, 且该模型具有低侵入、易实现、高可用、强扩展等优点.

关键词: 新型 SQL 注入攻击; 网络安全; HTTP Headers; 防御模型

Research and Defense of a New Type of SQL Injection Attack

ZHAO Yang, GUO Yu-Cui

(School of Science, Beijing University of Posts and Telecommunications, Beijing 100876, China)

Abstract: In this paper, a new type of SQL Injection attack through HTTP Headers is studied. Through analysising an example of the SQL Injection attack, the principle of the new type of SQL Injection attack is revealed, and the defense for the new type of SQL Injection attack is proposed. A defense model is established via such means as the IP filtering, data validation and machine learning, and this model has such advantages as low invasive, easy realization, high availability and strong expandability.

Key words: new type of SQL injection attack; network security; HTTP headers; defense model

SQL 注入攻击是一种应用广泛、具有很大威胁性的 Web 攻击技术,被列为 OWASP(Open Web Application Security Project) 十大 Web 应用系统安全威胁之首[1], 这种攻击普遍存在于许多 Web 应用程序中, 它时刻威胁着应用的数据安全[2]. 根据 Veracode 的 2014 应用程序的安全性和实践调查研究, SQL 注入漏洞仍然困扰着约 32%的 Web 应用程序[3]. 攻击者乐此不疲的原因是他们所攻击的目标数据库有着很强的吸引力, 数据库通常包含着应用中许多令人感兴趣的、有价值的数据[4]. 攻击者可以利用这个漏洞来修改数据库中的条目, 在数据库中执行命令(删除数据库, 更改权限等)以及读取数据库重要的数据信息, 从而给 Web 应用造成巨大的损失[5].

通常 SQL 注入主要通过 HTTP 请求 Web 服务器而进行, 在这种情况下的 SQL 攻击中, 输入数据会限定在 GET 和 POST 的变量上面[6], 因此大部分 SQL 注入的研究也集中在 GET 和 POST 上, 然而 HTTP 请求的 Header 的参数也可以被用来进行 SQL 注入攻击, 并且这种攻击手段更加隐蔽且难以发现, 其造成的后果却是巨大的, 难以估量的[7]. 以往的防御手段是通过减少漏洞或者规范编程来解决 SQL 注入的问题,但是黑客依然可以寻找新的攻击字符串来规避这些程序检测点[8-10].

文中重点分析了以 HTTP Header 为途径的新型 SQL 攻击的原理和发生场景, 并且针对这种新型攻击, 提出了一种新的有效的防御模型.

1 SQL注入攻击概述

SQL 注入攻击是指攻击者通过在应用程序的查询操作中插入一些 SQL 语句来操作数据. 具体来说, 它是利用现有应用程序, 通过在 Web 程序的用户输入中输入恶意 SQL 语句, 将其注入到存在安全漏洞的网站后台数据库引擎, 而后执行恶意的 SQL 命令, 而不是按照设计者意图去执行[11].

SQL 注入攻击的主要形式有两种: 一是直接将代码插入到与 SQL 命令串联在一起并使得其以执行的用

① 收稿时间:2015-10-06;收到修改稿时间:2015-11-27 [doi: 10.15888/j.cnki.csa.005195]

户输入变量, 由于其直接与 SQL 语句捆绑, 所以也被称为直接注入式攻击法[12]; 二是一种间接的攻击方法, 它将恶意代码写入要在数据库表中存储的字符串, 在存储的字符串中会连接到一个动态的 SQL 命令中, 以执行一些恶意的 SQL 代码[13].

2 HTTP Headers SQL注入

2.1 HTTP Headers 漏洞测试

HTTP 请求消息分为起始行, 头部(Header), 主体(Body)三部分, 每个部分均可以携带参数, 接下来对 HTTP 请求消息不同部分携带参数所包含的漏洞进行测试, 使用 90 多款市场上比较流行的漏洞扫描工具测试后得到结果如下图,

其中, HTTP Query String 参数(GET): 位于 URL 中; HTTP Body 参数(POST): 位于 HTTP body 中; HTTP Headers: 位于 HTTP 的 message header 中.

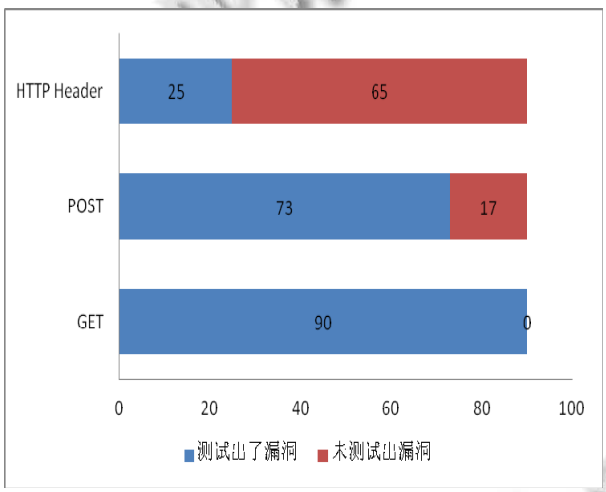

图 1 针对 HTTP 参数的扫描工具测试结果

通过该图 1 可以明显地看到, 72.2%的网络扫描工具没有测试出 HTTP Headers 中的漏洞. 此外, 相比于对 GET 和 POST 的测试情况, 这些扫描工具对 HTTP Headers 的检测能力令人堪忧, 因为 HTTP Headers 的漏洞具有很强的隐蔽性. 因此在实际应用中绝不应该忽视对该部分参数的漏洞检测.

2.2 HTTP Headers SQL 注入

容易产生漏洞的 HTTP Headers 的参数有 X-Forwarded-For, User-agent, Referer, Cookies 等, 下面以 Referer 参数为例子, 分析 HTTP Headers SQL(后文简称 HHS)注入攻击的原理.

Referer 是 HTTP Headers 的一部分, 当浏览器向 Web 服务器发送请求的时候, 一般会带上 Referer 字段, 记录服务器请求所来自的页面, 服务器借此可以获得一些信息用于数据处理. 比如从 A 主页上链接到 B 主页, 服务器就能够从 Referer 中统计出每天有多少用户点击 A 主页的链接访问 B 主页.

为了演示注入攻击, 搭建一个应用程序, 在数据库中存取了用户对应的 Referer, 用户登录系统后, 将会检查 HTTP 请求携带的 Referer 字段值, 如果它与存储在数据库中的值一致, 则该页面会重定向到一个购买页面 purchase.jsp, 页面的链接: http://localhost:8080/Test/headerSQLinjection/purchase.jsp.

正确请求携带的 Referer 值:

http://localhost:8080/Test/headerSQLinjection/.

应用数据库存储的 Referer 值:

http://localhost:8080/Test/headerSQLinjection/.

很明显正常情况下, HTTP 请求携带的 Referer 字段值总是与应用中存储值相匹配. 假设如下情况, 攻击者在其恶意网页里设置一个链接 purchase.jsp, 点击该连接时, 攻击者将能够代表点击者执行动作, 在应用中扮演点击者的角色进行买卖.

接下来从攻击者的角度分析, 攻击者渗入到网站, 进入管理面板, 找到 purchase.jsp, 复制 purchase.jsp 链接并将其粘贴在浏览器中, 会发现攻击者并未成功进入, 因为如果是正常登陆, 系统通过匹配 Referer 报头, 使得重定向到 purchase.jsp, 而直接复制链接, 因为 Referer 报头不匹配, 从而无法进入.

第一步, 发起一个正常的请求, 如下:

POST / HTTP1.1

Host:localhost

User-Agent:Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.04506.648; .NET CLR 3.5.21022)

Referer: http://localhost:8080/Test/headerSQLinjection/

Content-Type:application/www-form-urlencoded

Content-Length:40

Connection: Keep-Alive

UserName=root&Pw=root&Submit=Submit

其返回的响应为:redirect.jsp

<html>

<head><title>重定向页面</title></head>

<body><h3>5 秒之后将会被重定向</h3>
<script>setTimeout(windows.location='http://localhost:8080/Test/headerSQLinjection/purchase.jsp',5000);</script>
</body>
</html>

第二步, 改变请求中的 Referer 字段, 将其改为 http://example.com, 其返回的响应 error.jsp

<html>
<head><title>错误页面</title></head>
<body>
<h3>该请求不是一个合法的来源</h3>
</body>
</html>

虽然请求失败了, 但是上述返回的响应展示了两个有用的信息:

a.应该存在一个合法的 Referer 报头.

b.Referer 储存在应用数据库中, 用来检查用户 http 请求 headers 的 Referer 是否与数据库中的一致. 这就表明应用为了再次检查 Referer, 数据库是参与其中的, 这样就找到了针对 HTTP Headers 的 SQL 注入点.

第三步, 将请求中的 Referer 字段, 改为: http://localhost:8080/Test/headerSQLinjection/'OR(select 'abcD' from DUAL where 1=1 and SLEEP(5))OR';

DUAL 是 MySql 的一个虚拟表, 当数据不是来自创建表中的时候, 可以使用该虚拟表, 因为不知道应用的数据库表名, 所以可以使用此虚拟表来达到访问数据库的目的. OR(select 'abcD' from DUAL where 1=1 and SLEEP(5))OR 表明查询需要等待 5 秒, 5 秒后页面回应, 此时通过页面反馈的信息很容易判断出是否容易受到 SQL 注入攻击.

3 HHS注入攻击的防御模型

本文针对 HHS 注入攻击的隐蔽性, 多样性的特点, 设计了如下的防御模型, 该防御模型主要分三层: 校验层, 持久层, 源数据层, 其中校验层包含 IP 过滤模块, 可疑字符校验模块, 一致性校验模块, 攻击源定位模块; 持久层包含数据存取模块, 主从数据库; 源数据层由 Web 服务器的数据库组成. 校验层和持久层为该防御模型的重点部分, 该防御系统模型如图 2 所示.

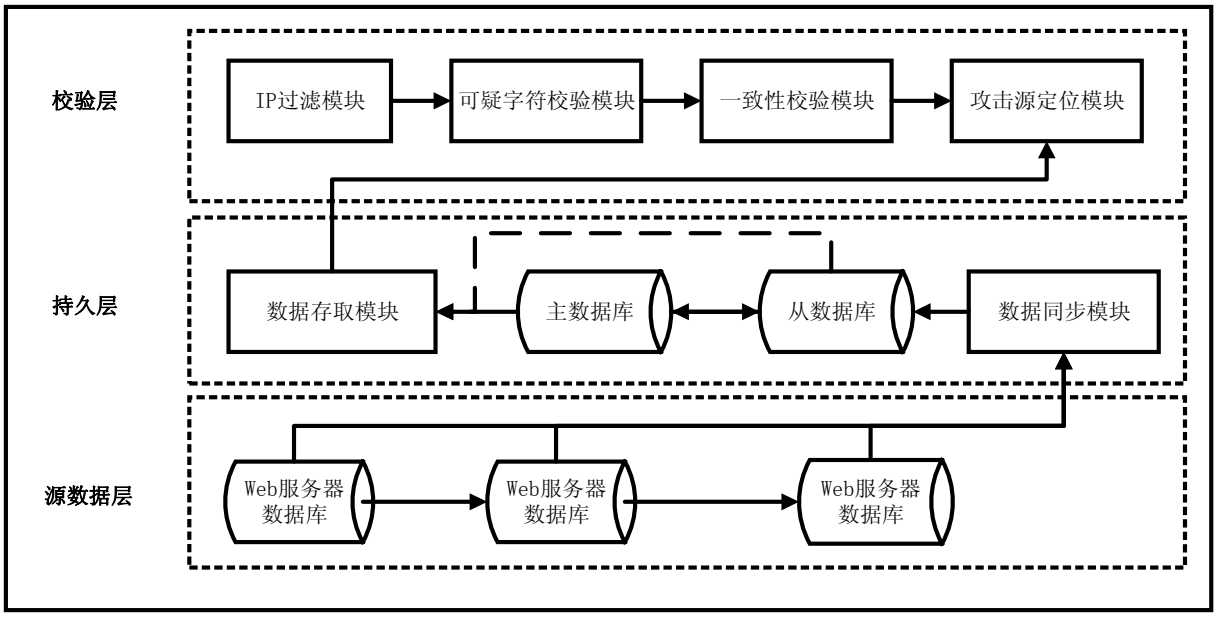

图 2 系统模型图

3.1 防御模型的层次及模块介绍

3.1.1 校验层

该层主要由 IP 过滤模块, 可疑字符校验模块, 一致性校验模块, 攻击源定位模块组成, 该层实现了防御模型的主要功能, 该层的设计必须是可扩展, 可配置的.

① IP 过滤模块

该模块维护着一个 IP 地址库, 该地址库存储着一些持续对系统进行 SQL 注入攻击的 IP 地址, 模块根据地址库的信息对非法请求的 IP 地址进行封禁, 从而使得攻击者的非法请求被阻止, 该模块的 IP 地址库不定时更新, 更新来源是攻击源定位模块. 当攻击源定位模块确定攻击者的 IP 地址后会将其更新到该模块.

② 可疑字符校验模块

该模块维护着一个可疑字符串库, 库里所包含的内容随着新的可疑字符串的出现, 动态更新, 该模块有自我学习的机制. 通过对 HTTP headers 中参数进行分析, 查找参数所对应的内容中是否包含可疑字符库中的数据, 采用白名单机制, 如果不包含则认为参数是可靠的, 请求是可以继续进行的, 之所以不采用黑名单机制, 是因为黑名单机制是如果包含某些可以字符就禁止, 这样有可能增加误报的几率, 使得合法的请求不能进行.

③ 一致性校验模块

该模块首先从 Web 服务器查找出存储在其数据库中正确的 HTTP headers 参数, 然后将用户 HTTP 请求中 headers 的参数分别与数据库中的进行比对. 从而确定客户端传来的 HTTP 请求所携带的 headers 参数是否经过篡改.

④ 攻击源定位模块

攻击源定位模块维持着一个计数器, 初始化均为 0, 经过可疑字符串校验模块校验后, 若 HTTP 请求存在可疑字符串, 则攻击源定位模块首先定位此 HTTP 请求的 IP 地址, 然后将其对应的计数加一, 经过一致性校验模块的过滤后, 若出现不一致, 则攻击源定位模块针也会对此 HTTP 请求的 IP 地址的计数进行加一操作, 并通过数据存取模块将计数结果持久化到数据库.

3.1.2 持久层

该层主要作用有如下两个:

① 是从下游源数据层的 Web 服务器数据库中获取与 HTTP headers 相关数据表的数据, 并将数据存储到该层的数据库中.

数据同步模块实现该作用, 它保证了持久层数据库的数据与 Web 服务器数据库数据的一致性. 校验层之所以不直接从 Web 服务器数据库中提取数据, 而是中间增加了一个持久层, 是因为如果直接从 Web 服务器数据库中获取数据一方面会增加 Web 服务器数据库的访问次数, 增加了数据库出现 IO 瓶颈的几率, 另一方面如果直接访问 Web 服务器的数据库增加了其被直接攻击的可能性, 攻击者可能会利用防御模型的缺陷对 Web 服务器数据库进行攻击.

② 是为上游校验层的一致性校验模块和攻击源定位模块提供原始数据, 同时为攻击源定位模块的计数结果提供持久化存储, 便于为以后分析提供历史数据.

数据存取模块实现了该需求, 它对外提供了读写数据库的功能, 为校验层的一致性校验模块提供了 HTTP headers(referer, user-agent, cookie 等)的原始准确数据; 同时它为校验层的攻击源定位模块提供了历史数据的读取和新增数据的写入功能.

3.1.3 源数据层

该层主要由 Web 服务器端的数据库组成, 在部署防御模型时, 应该对该层的数据库进行权限设置, 仅把数据库中与 HTTP headers 相关的数据表对持久层开放, 并且只开放读取操作, 不开放写入操作, 以防止由于防御模型的漏洞导致该层数据库的数据被篡改.

3.2 防御模型的工作流程

该防御模型的工作流程如图 3 所示.

具体流程如下:

① 用户在客户端提交 HTTP 请求进入 IP 过滤模块, 若该请求的 IP 地址在 IP 过滤模块的禁用 IP 地址库中, 则进行步骤②. 若请求的 IP 地址未在禁用 IP 地址库中, 则进行步骤③;

② 请求被拒绝, 提示用户由于有非法操作历史, IP 被封禁, 导致无法进行请求, 用户可以选择申诉或者等待解禁;

③ 请求进入可疑字符校验模块, 该模块对请求的 headers 参数进行校验, 检查参数中是否存在非法字符, 若没有则进行步骤④, 若存在非法字符则进行⑤;

④ 请求进入一致性校验模块, 从数据库中查找对应的参数与 headers 的参数进行比对. 如果两者一致, 则进行步骤⑥, 不一致则进行步骤⑤;

⑤ 攻击源定位模块的定位请求的 IP 地址, 并将其计数进行加一操作, 然后进行步骤⑦;

⑥ 正常的对 Web 服务器进行请求, 将响应返回给客户端;

⑦ 判断出请求为 SQL 注入攻击, 请求被返回到客户端, 用户需要重新进行步骤①,同时如果请求的 IP 地址对应的计数大于阈值, 则进行⑧;

⑧ 将请求的 IP 地址更新到 IP 过滤模块的禁用 IP

地址库同时进行步骤②.

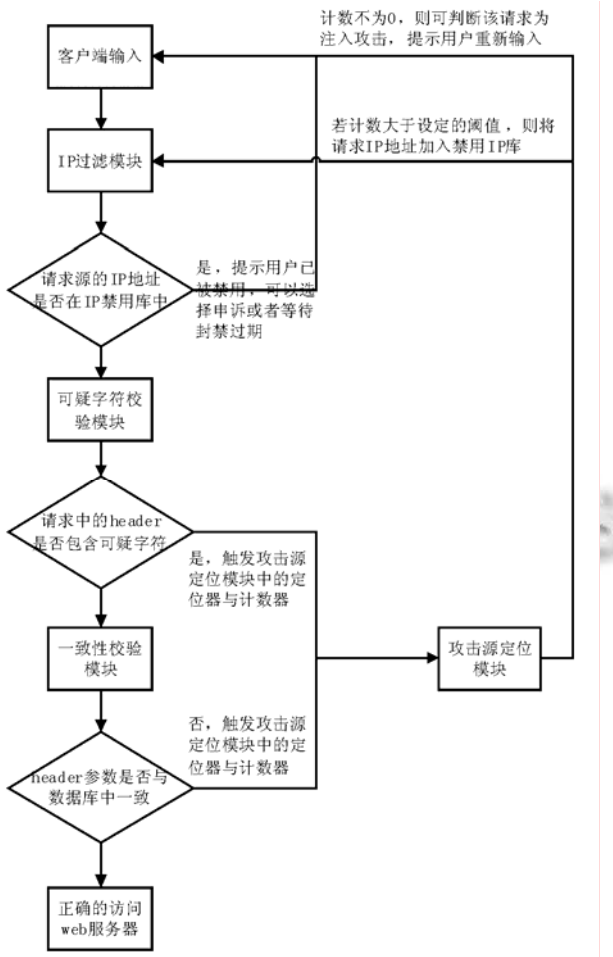

图 3 防御模型的工作流程图

3.3 防御模型的优势

① 扩展性强, 灵活地进行新功能的增加

由于校验层有对外的接口, 当需要进行扩展时, 只需在校验层新增模块便可, 比如, 当需要增加对其他类型的 SQL 注入攻击校验的模块, 只需要在可疑字符校验模块的前面加一个新的模块, 该模块可以是一个已有的防御系统也可以是新写的校验模块, 具有很大的灵活性.

② 主从备份, 灾备

针对持久层的数据, 该模型设计了主从备份, 保证了系统的高可用, 灾备性.

③ 无需修改 Web 服务网站的源代码

通过系统模型图可以看出, 当需要将该防御模型应用到 wed 服务中的时候, 只需将 Web 服务器的数据库中 HTTP headers 相关的数据表同步到模型持久层的数据库中便可, 这个过程只需要执行几条 SQL 查询语句, 无需对 Web 服务网站源代码进行修改, 这样极大的简化了该防御模型接入 Web 网站的过程. 降低了双方的耦合度, 有利于快速部署和版本迭代.

4 实验结果分析

4.1 实验环境

搭建一个简单的 SQL 注入攻击实验环境, 需要两台 Linux 主机, 一台作为攻击者(Ubuntu 系统), 上面运行着 SQLMap 程序, 作为进行注入攻击的工具, 一台运行着包含注入漏洞的四个 web 应用, 搭载着 MYSQL 数据库, 作为攻击目标. 攻击者的 IP 地址设置为 192.168.1.101, 攻击目标服务器 IP 地址设为 192.168.1.102. SQLmap 是一款用来检测与利用 SQL 注入漏洞的免费开源工具, 通过配置 SQLMap 的参数对 web 应用发起不同的 HTTP 请求, 这些请求中有正常输入请求, 有恶意输入请求.

4.2 实验结果分析

对 web 应用进行大量的 HTTP 请求发现, 本文所提出的 SQL 防御模型针对普通的以 get, post 方式进行的 SQL 注入请求有百分百的防御准确率, 如下表 1 所示, 针对以 HTTP Header 为途径的 SQL 注入请求有百分九十八的防御准确率, 之所以出现漏报, 经分析是因为防御模型中的可疑字符校验模块中的数据集不完整导致, 因为该模块需要长时间的学习才能获得较为完整的可疑字符. 该系统针对以 HTTP Header 为途径的 SQL 注入存在约 1.8%误报率, 究其原因是某些正常输入包含可疑字符, 非法字符, 防御系统直接将其视为恶意攻击.

其中, 表格中准确率, 误报率, 以逗号分割, 前半部分为 get, post 注入攻击请求的数据, 后半部分为 httpheader 注入攻击请求的数据.

表 1 防御性能测试结果

应用	请求数	Get post 注入请求数	HHS 注入请求数	准确率(%)	误报率(%)
A	500	300	108	100, 97.8	0,2.3
B	500	218	254	100, 98.4	0,1.8
C	500	200	204	100, 97.6	0,2.5
D	500	281	112	100, 99 .2	0,1.1

因为在 web 应用上加载了防御系统, 因此对于原来的访问请求会造成一定的响应延迟. 该防御系统对

请求响应的影响如下表 2 所示, 可见由于需要进行一些列的校验, 造成请求响应时间变长, 对用户体验有一定的影响, 这是无法避免的, 不过可以通过对校验层进行异步校验, 缩短响应时间, 使用户使用起来没有明显的卡顿, 这也是需要进行进一步研究的地方.

表 2 响应时间测试结果

应用	请求数	无防御系统每个请求的平均响应时间	有防御系统每个请求的平均响应时间
A	500	1.3	2.5
B	500	1.5	3.2
C	500	0.9	1.9

5 总结

针对互联网上的 Web 服务网站进行的 SQL 注入攻击, 仍然具有顽强的生命力, 它仍然困扰着许多 Web 应用, 因此针对这个方向的研究应该随着互联网的发展而跟进, 针对一些新型的攻击手段应该及时提出可靠的防御模型.

虽然本文所设计的防御模型对 HHS 注入有不错的防御效果, 但仍有一些地方需要进一步改进和完善, 可以从如下几个方面进行扩展研究:

① 防御模型的全面性:该防御系统仅对 HHS 注入攻击进行了防御, 但是一些其他的攻击手段并未进行考虑, 虽然可以通过在该模型的上游或者下游进行扩展, 增加针对其他攻击手段的过滤, 但是这无疑增加了请求的访问速度, 会给用户体验带来不良的效果.

② 防御模型的鲁棒性: 攻击者可能会针对系统本身进行攻击, 使该防御系统瘫痪或者出现误报, 导致正常的请求无法进行, 从而达到攻击 Web 网站的最终目的. 可以针对防御模型进行更加细致的设计, 通过加密, 代理等手段预防攻击, 从而提高防御系统的鲁棒性.

③ 防御模型 的的智能化:由于攻击源定位模块还不具备足够的智能化, 可能会有漏报, 误报的发生, 近年来机器学习已经在业界得到广泛应用, 可以利用机器学习的方法, 对攻击者所具有的特征进行数据挖掘, 通过训练数据集, 测试数据集等方法来提高该模块检验机制的准确性.

参考文献

1 钱丽.SQL 注入攻击原理及其防范技术研究.哈尔滨师范大学自然科学学报,2014,6(30):66−68.

2 田玉杰,赵泽茂,张海川,李学双.二阶 SQL 注入攻击防御模型.信息网络安全,2014,11(12):70−73.

3 张成.SQL 注入攻击防范技术研究.安徽科技学院学报,2015,29(2):63−65.

4 胡博.WEB 应用的 SQL 注入攻击和防范技术研究.电脑知识与技术,2013,9(33):7408−7412.

5 冯谷,高鹏.新型 SQL 注入技术研究与分析.计算机科学,2012,11(39):415−423.

6 张楠,张振国.基于规则的检测 SQL 注入攻击方法的研究.陕西科技大学学报,2014,2(25):121−123.

7 郜激扬.基于 Web 服务的数据库注入攻击与防范.华北水利水电学院学报,2008,1(29):89−91.

8 Patel N, Mohammed F, Soni S. SQL injection attacks: Techniques and protection mechanisms. International Journal on Computer Science and Engineering, 2011, 1(3): 199−203.

9 Das D, Sharam U, Bhattacharyya DK. An approach to detection of SQL injection attack based on dynamic query matching. International Journal of Computer Applications, 2010, 25(1): 28−33.

10 陈锦.Oracle 数据库 SQL 注入技术研究.信息网络安全,2011,12(22):80−83.

11 谢希仁.计算机网络(第 5 版).北京:电子工业出版社,2011:100−160.

12 王云,郭外萍,陈承欢.Web 项目中 SQL 注入问题研究与方法防范.信息安全技术,2010,31(5):976−978.

13 车延雪,王志强.SQL 注入式攻击与防范.网络安全技术与应用,2014,11(2):98−91.